Cardinality Estimation Using Gumbel Distribution

Aleksander Łukasiewicz
Faculty of Mathematics and Computer Science, University of Wrocław, Poland

Przemysław Uznański
Faculty of Mathematics and Computer Science, University of Wrocław, Poland

Abstract

Cardinality estimation is the task of approximating the number of distinct elements in a large dataset with possibly repeating elements. LogLog and HyperLogLog (c.f. Durand and Flajolet [ESA 2003], Flajolet et al. [Discrete Math Theor. 2007]) are small space sketching schemes for cardinality estimation, which have both strong theoretical guarantees of performance and are highly effective in practice. This makes them a highly popular solution with many implementations in big-data systems (e.g. Algebird, Apache DataSketches, BigQuery, Presto and Redis). However, despite having simple and elegant formulation, both the analysis of LogLog and HyperLogLog are extremely involved – spanning over tens of pages of analytic combinatorics and complex function analysis.

We propose a modification to both LogLog and HyperLogLog that replaces discrete geometric distribution with the continuous Gumbel distribution. This leads to a very short, simple and elementary analysis of estimation guarantees, and smoother behavior of the estimator.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Streaming algorithms, Cardinality estimation, Sketching, Gumbel distribution

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.76

Related Version *Full Version*: https://arxiv.org/abs/2008.07590

Funding *Aleksander Łukasiewicz*: Polish National Science Centre grant 2019/33/B/ST6/00298.
Przemysław Uznański: Polish National Science Centre grant 2019/33/B/ST6/00298.

Acknowledgements We would like to thank Seth Pettie for useful remarks that helped us improve the paper.

1 Introduction

In the cardinality estimation problem we are presented with a dataset consisting of many items, and some of these items might appear more than once. Our goal is to process this dataset efficiently, in order to estimate the number n of *distinct* elements it contains. Here, efficiently means in small auxiliary space, and with fast processing time per each item. A natural scenario to consider is a *stream* processing of a dataset, with stream of events being either element *insertions* to the multiset or *queries* of the multiset cardinality.

A folklore information theoretic analysis reveals that this problem over universe of u elements requires at least u bits of memory to answer queries exactly. However, in many practical settings it is sufficient to provide an approximation of the actual cardinality. One of the possible real-world scenarios is a problem of estimating the number of unique addresses in packets that a router observes, in order to detect malicious behaviors and attacks. Here, the challenge arises from the limited computational capabilities of the router and sheer volume of the data that can be observed over e.g. a day.

The theoretical study of the *cardinality estimation* was initiated by the seminal work of Flajolet and Martin [20]. From that point, two separate lines of research follow. First, there has been a considerable effort put into developing approximation schemes with so called (ε, δ)-guarantees, meaning that they guarantee outputting $(1+\varepsilon)$-multiplicative approximation of the cardinality, with probability at least $1 - \delta$. Here, we mention [6, 7, 8, 11, 22, 23, 29] on the upper-bound side and [6, 10, 27, 28, 36] on lower-bound side. The high-level takeaway message is that one can construct approximate schemes that provide $(1 + \varepsilon)$-multiplicative approximation to the number of distinct elements, using an order of ε^{-2} space, and that this dependency on ε is tight. More specifically, the work of Błasiok [11] settles the bit-complexity of the problem, by providing $\mathcal{O}(\frac{\log \delta^{-1}}{\varepsilon^2} + \log n)$ bits of space upper-bound, and this complexity is optimal by a matching lower bound [28]. To achieve such small space usage, a number of issues have to be resolved, and a very sophisticated machinery of expanders and pseudo-randomness is deployed.

The other line of work is more practical in nature, and focuses on providing variance bounds for efficient algorithms. The bounds are usually of the form $\sim 1/\sqrt{k}$ where k is some measure of space-complexity of algorithms (usually, corresponds to the number of parallel estimation processes). This approach includes work of [9, 12, 14, 16, 18, 19, 21, 24, 30, 31, 34, 35]. Recently, Pettie and Wang started the study of the intrinsic tradeoff between the space complexity of the cardinality estimation sketch and its estimation error by introducing the notion of *memory-variance product* (MVP) [31]. They proposed a Fishmonger sketch that has an MVP equal to $H_0/I_0 \approx 1.98$ (where H_0, I_0 are some precisely defined constants) and they also proved that this is the best MVP that one can get in a class of *linearizable* sketches (in fact all the popular *mergeable* sketches are linearizable). In the very recent follow-up work Pettie, Wang and Yin studied the MVPs of non-mergeable sketches [32].

We now focus on two specific algorithms, namely LogLog [16] and its refined version called HyperLogLog [19]. The guarantees that these algorithms provide for variance are approximately $1.3/\sqrt{k}$ and $1.04/\sqrt{k}$ respectively, when using k integer registers. Both are based on a simple principle of observing the maximal number of trailing zeroes in the binary representation of hashes of elements in the stream, although they vary in the way they extract the final estimate from this observed value (we will discuss those details in the following section). In addition to being easy to state and provided with theoretical guarantees, they are highly practical in nature. We note the following works on algorithmic engineering of practical variants [17, 26, 37], with actual implementations e.g. in Algebird [1], BigQuery [2], Apache DataSketch [3], Presto [4] and Redis [5].

Despite its simplicity and popularity, LogLog and HyperLogLog are exceptionally tough to analyze. We note that both papers analyzing LogLog and later HyperLogLog use a heavy machinery of tools from analytic combinatorics and complex function analysis e.g. Mellin transform, poissonization and analytical depoissonization. In fact, unpacking the main tool used in the paper requires understanding of another tens of pages from [33]. Additionally, both papers are presented in a highly compressed form. Thus, the analysis is not easily digestible by a typical computer scientist, and has to be accepted “as it is” in a black-box manner, without actually unpacking it.

This creates an unsatisfactory situation where one of the most popular and most elegant algorithms for the cardinality estimation problem has to be treated as a black-box from the perspective of its performance guarantees. It is an obstacle both in terms of popularization of the LogLog and HyperLogLog algorithms, and in terms of scientific progress. We note that those algorithms are generally omitted during majority of theoretical courses on streaming and big data algorithms.

Our contribution: Gumbel distribution

Our contribution comes in two factors. First, we observe that the key part of LogLog and HyperLogLog algorithms is counting the trailing zeroes in the binary representation of a hash of element. This random variable is distributed according to a geometric distribution. Both

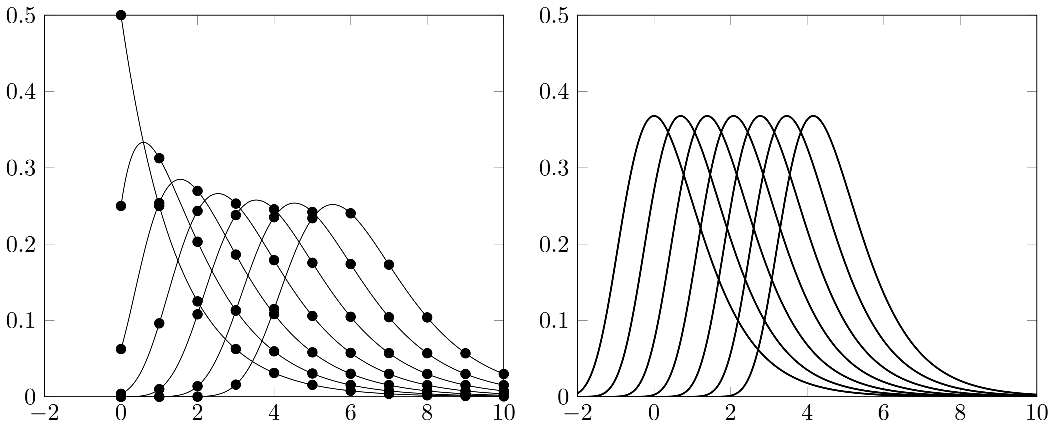

Figure 1 Distribution of $\max\{X_1, \ldots, X_k\}$ for $k \in \{1, 2, 4, 8, 16, 32, 64\}$ where X_i iid random variables distributed according to discrete Geometric distribution (on the left) and Gumbel distribution (on the right). Discrete distribution given by $f_k(x) = (1-2^{-x-1})^k - (1-2^{-x})^k$ is drawn with continuous intermediate values for smooth drawing.

LogLog and HyperLogLog estimate the cardinality using the maximum value of the count of trailing zeroes observed over all elements of the dataset. However, the distribution of the maximum of many discrete random variables drawn from an identical geometric distribution is not distributed according to a geometric distribution. This is unwieldy to handle in the analysis in [19].

We propose the following: as the first step we replace the discrete geometric distribution with its continuous counterpart, i.e. the exponential distribution with the CDF $1 - e^{-x}$. Next, we take a maximum of N independent repetitions of our algorithm which can be simulated by, e.g., replacing each update x with N updates of the form (x, i) for $i \in [N]$. This yields the CDF of the form $(1 - e^{-x})^N$. Intuitively, we expect this manipulation to have a smoothing effect on the irregularities of LogLog and HyperLogLog (which performance deteriorate greatly for very small values of n). Third and the final step is to take a limit of $N \to \infty$, while maintaining a proper normalization of the distribution (i.e., we take a shift by $\ln N$), resulting in a CDF of the form $F(X) = \lim_{N\to\infty}(1 - e^{-x-\ln N})^N$.

A little manipulation gives us $F(X) = \lim_{N\to\infty}(1 - \frac{e^{-x}}{N})^N = e^{-e^{-x}}$ which is precisely the CDF of the Gumbel distribution, with the following crucial property

If $X_1, \ldots, X_k$ are independent random variables drawn from a Gumbel distribution, then $Z = \max(X_1, \ldots, X_k) - \ln(k)$ is also distributed according to the same Gumbel distribution.

This allows us to simplify extraction of the value of k from $\max(X_1, \ldots, X_k)$, since we are always dealing with the same type of error (distributed according to the Gumbel distribution) on top of the value $\ln(k)$.

Our contribution: Simpler analysis

Our second contribution comes in the form of a simple analysis of the performance guarantees of the estimation. We note that since our observable can be interpreted as an observable from LogLog or HyperLogLog repeated N-times (for some very large value of N), we expect to get a similar type of the guarantees. One should be able to go with the tour-de-force analysis analogous to [16, 19]. However, we find it valuable to provide analysis that is tractable using just elementary and short proofs. We show that by taking advantage of the Gumbel distribution being the limiting distribution, one can isolate a very simple combinatorial problem capturing the essence of the stochastic averaging. The analysis of this problem requires application of only some basic probabilistic inequalities and multinomial identities.[1]

2 Related work

The key concept used in virtually all cardinality estimation results, can be summarized as follows: given a universe U of elements, we start by picking a hash-function. Then, given a subset $M \subseteq U$ which cardinality we want to estimate, we proceed by applying h to every element of M and operate only on $M' = \{h(x) : x \in M\} \subset [0,1]$. The next step is computing an *observable* – i.e. a quantity that only depends on the underlying set and is independent of replications. In the final step we somehow extract the estimate of the cardinality from the observable.

For example [7] uses $h : M \to [0,1]$ and the value $y = \min M' = \min_{x \in M} h(x)$ as an observable. We expect $y \sim \frac{1}{n+1}$, thus $\frac{1}{y} - 1$ is used as an estimate of the cardinality n. However, since we need to overcome the variance, one might need to average over many independent instances of the process, in order to achieve a good estimation. In this particular example, to get an $(1+\varepsilon)$ approximation, we need to average over $\mathcal{O}(\varepsilon^{-2})$ independent copies of the algorithm. Therefore, the total memory usage becomes $\mathcal{O}(\varepsilon^{-2} \log n)$ bits.

Stochastic averaging

Naively averaging over k independent copies of the algorithm has an important drawback - the time for processing each query grows from $O(1)$ to $O(k)$. *Stochastic averaging* is a technique designed to address that issue. In our setting it works as follows: instead of processing each element in each of the k processes independently (which is a bottleneck), we randomly partition our input into k disjoint sub-inputs: $M = M_1 \cup \ldots \cup M_k$, and we run each copy of an algorithm only on its corresponding sub-input. This is simulated by picking a second hash function $h' : M \to \{1, \ldots, k\}$, and when we are processing an element x, it is assigned to M_i where $i = h'(x)$ is decided solely on the hash of x. Intuitively, we expect each M_i to contain roughly n/k elements. Note that the actual number of elements in all M_i follows a *multinomial distribution*, and this presents an additional challenge in the analysis.

LogLog sketching

Consider the following: we hash the elements to bitstrings, that is $h : M \to \{0,1\}^\infty$, and consider the bit-patterns observed. For each element find the value $\mathsf{bit}(x)$ such that $h(x)$ has a prefix $0^{\mathsf{bit}(x)}1$. The particular value $\mathsf{bit}(x) = c$ should be observed once every $\sim 2^c$ different hashes, and can be used to estimate the cardinality. The observable used in the LogLog is the value $\max_x \mathsf{bit}(x)$ among all elements. Since we expect its value to be roughly of order $\log n$, we maintain the value of $\max \mathsf{bit}(x)$ on $\mathcal{O}(\log \log n)$ bits.

Denote the observables produced in the concurrent copies of the algorithm as $t_1, \ldots, t_k$. We expect the values of t_i to be such that $2^{t_i} \sim n/k$. One can easily show, that for any t_i, we have $\mathbb{E}[2^{t_i}] = \infty$, thus taking the arithmetic average over 2^{t_i} is not a feasible strategy.

[1] It is important to note that this is not the **first** cardinality estimation algorithm with a simple analysis, e.g. [7, 20] algorithms have relatively straightforward analysis. However, none of those techniques apply to the simplification of LogLog or HyperLogLog specifically, which are default practical choices for the cardinality estimation.

However, it turns out that the geometric average works in this setting, and we expect the $k\left(\prod_i 2^{t_i}\right)^{1/k}$ to be an estimate for n (one also needs a normalizing constant that depends solely on k). The analysis in [16] shows that the variance of the estimation is roughly $1.3/\sqrt{k}$.

HyperLogLog sketching

HyperLogLog ([19]) is an improvement over LogLog with the observation that the *harmonic average* achieves a better averaging performance over *geometric average*. Thus HyperLogLog is constructed by setting the estimator to $k^2\left(\sum_i 2^{-t_i}\right)^{-1}$ with some normalizing constant (depending on k). The resulting algorithm has a variance which is roughly $1.04/\sqrt{k}$.

In fact it can be shown that the harmonic average is optimal in that setting: among observables that constitute of taking maximum of a hash function, harmonic average is a *maximum likelihood estimator* (see e.g. [13]). However, this claim is strict only without stochastic averaging.

Due to the space limitations, in this article we provide only the analysis of the LogLog version (with geometric average estimation) of our algorithm. The HyperLogLog version (using harmonic average estimation) is available in the full version of the paper. [2]

3 Preliminaries

Computational model

We assume oracle access to a perfect source of randomness, that is a hash function $h : [u] \to \{0,1\}^\infty$. If the sketch demands it, we allow it to access multiple independent such sources, which can be simulated with a help of bit and arithmetic operations. The oracle access is a standard assumption in this line of work (c.f. discussion in [31]) – the purpose of this assumption is to separate the analysis of the space complexity of the algorithm from the space complexity of the source of the randomness.

Besides that, we assume standard RAM model, with words of size $\log u$ and standard arithmetic operations on those words taking constant time.

Gumbel distribution

We use the following distribution, which originates from the *extreme value theory*.

▶ **Definition 1** (Gumbel distribution [25]). *Let* $\mathsf{Gumbel}(\mu)$ *denote the distribution given by a following CDF:*

$$F(x) = e^{-e^{-(x-\mu)}}.$$

Its probability density function is given by

$$f(x) = e^{-e^{-(x-\mu)}} e^{-(x-\mu)}.$$

Observe that, directly from the definition, if $X \sim \mathsf{Gumbel}(\mu)$, then $X+c \sim \mathsf{Gumbel}(\mu+c)$.

We also note that when $x \to \infty$, then $f(x) \approx e^{-(x-\mu)}$, thus the Gumbel distribution has an exponential tail on the positive side. The distribution has a doubly-exponential tail when $x \to -\infty$.

[2] The full version of the paper is available under the following link: `https://arxiv.org/abs/2008.07590`.

We have the following basic property when $X \sim \mathsf{Gumbel}(\mu)$ (c.f. [25]):

$$\mathbb{E}[e^{\alpha X}] = e^{\alpha\mu} \int_{-\infty}^{\infty} e^{-e^{-x}} e^{(\alpha-1)x} dx = e^{\alpha\mu}\Gamma(1-\alpha), \tag{1}$$

from which it follows that $\mathbb{E}[e^{-X}] = e^{-\mu}$ and $\mathrm{Var}[e^{-X}] = e^{-2\mu}$.

▶ **Property 2** (Sampling from Gumbel distribution.)**.** *If $t \in [0, 1]$ is drawn uniformly at random, then $X = -\ln(-\ln t) + \mu$ has the distribution* $\mathsf{Gumbel}(\mu)$*.*

The following property is the key property used in our algorithm analysis. It essentially states that Gumbel distribution is invariant under taking the maximum of independent samples (up to normalization).[3]

▶ **Property 3.** *If $x_1, x_2, \ldots, x_n \sim \mathsf{Gumbel}(0)$ are independent random variables, then for $Z = \max(x_1, \ldots, x_n)$ we have $Z \sim \mathsf{Gumbel}(\ln n)$.*

Proof.

$$\Pr(Z < x) = \prod_i \Pr(x_i < x) = (e^{e^{-x}})^n = e^{e^{-x+\ln n}}.$$ ◀

Multinomial distribution

We now discuss the multinomial distribution and its role in the analysis of the stochastic averaging.

▶ **Definition 4.** *We say that $X_1, \ldots, X_k$ are distributed according to Multinomial$(n; p_1, \ldots, p_k)$ distribution for some $\sum_i p_i = 1$, if, for any $n_1 + \ldots + n_k = n$ there is*

$$\Pr[X_1 = n_1 \wedge \ldots \wedge X_k = n_k] = \binom{n}{n_1, \ldots, n_k} p_1^{n_1} \ldots p_k^{n_k}.$$

Consider a process of distributing n identical balls to k urns, where for each ball we place it in the urn i with probability p_i, fully independently between the balls. Then, the vector of the total number of balls in each urn $X_1, \ldots, X_k$ follows $\mathsf{Multinomial}(n; p_1, \ldots, p_k)$ distribution.

For our purposes we are interested in the following setting: let f be some real-valued function. Lets say that we have a stochastic process of estimating cardinality in a stream, that is if n distinct elements appear, the process outputs a value that is concentrated around its expected value $f(n)$. Now, we apply stochastic averaging, by splitting the stream into sub-streams, and feed each sub-stream to estimation process separately, say n_i going into sub-stream i. We can look at the following random variables:

$$S_n = \mathbb{E}[\sum_i f(n_i)] \qquad \text{and} \qquad P_n = \mathbb{E}[\prod_i f(n_i)].$$

[3] In fact, the Fisher–Tippett–Gnedenko theorem (c.f. [15]) states, that for any distribution $\mathcal{D}$, if for some a_n, b_n the limit $\lim_{n\to\infty}(\frac{\max(X_1,\ldots,X_n)-b_n}{a_n})$ converges to some non-degenerate distribution, where $X_1, \ldots X_n \sim \mathcal{D}$ (and are independent), then it converges to one of three possible distribution families: a Fréchet distribution, a Weibull distribution or a Gumbel distribution. Thus, those three distributions can be viewed as a counterpart to normal distribution, w.r.t. to taking maximum (instead of repeated additions).

We expect $S_n \approx kf(n/k)$ and $P_n \approx f(n/k)^k$. Deriving actual concentration bounds for specifically chosen functions f gives us insight on how well harmonic average or geometric average performs when concentrating cardinality estimation processes under stochastic averaging.

The analysis of the stochastic averaging for a *generic* function f (under some regularity constraints) has been done in [13]. We actually derive a stronger set of bounds for very specific functions: $f(x) = \ln(x+1)$ and $f(x) = \frac{1}{x+1}$.

4 Geometric average estimation

We start by showing a simple concentration result for geometric average of independent random variables distributed according to the Gumbel distribution.

▶ **Lemma 5.** *Let $G_1, \ldots, G_k$ be independent random variables distributed according to Gumbel(0), and let $G = \sum_i G_i$. If $k > 1$, then $\mathbb{E}[\exp(G/k)] = \Gamma(1-1/k)^k = \exp(\gamma)(1 + \frac{\pi^2}{12}\frac{1}{k} + \mathcal{O}(k^{-2}))$. If $k > 2$, then $\mathrm{Var}[\exp(G/k)] = \Gamma(1-2/k)^k - \Gamma(1-1/k)^{2k} = \exp(2\gamma) \cdot \left(\frac{\pi^2}{6k} + \mathcal{O}(k^{-2})\right)$*

Proof.

$$\begin{aligned}
\mathbb{E}[\exp(G/k)] &= \prod_i \mathbb{E}[\exp(G_i/k)] = \Gamma(1-1/k)^k \\
\mathrm{Var}[\exp(G/k)] &= \prod_i \mathbb{E}[\exp(G_i/k)^2] - \prod_i \mathbb{E}[\exp(G_i/k)]^2 \\
&= \Gamma(1-2/k)^k - \Gamma(1-1/k)^{2k}.
\end{aligned}$$

From the Taylor expansion of the log-gamma function we get that $\Gamma(1-z) = \exp(\gamma z + \frac{\pi^2}{12} z^2 + O(z^3))$, which yields the desired approximations. ◀

The following algorithm shows that if we are fine with slower updates, then the Gumbel distribution fits nicely into the standard cardinality estimation framework. The main idea is just to hash each element into a real-value distributed according to Gumbel distribution, and take the maximum across the values.

Algorithm 1 Cardinality estimation using Gumbel distribution.

Procedure *Init()*
  pick $r_1, \ldots, r_k : U \to [0,1]$ as independent hash functions
  $X_1 \leftarrow -\infty, \ldots, X_k \leftarrow -\infty$
Procedure *Update*(x)
  for $1 \le i \le k$ **do**
    $v \leftarrow -\ln(-\ln r_i(x))$ `// Gumbel(0) RV`
    $X_i \leftarrow \max(v, X_i)$
Procedure *GeometricEstimate()*
  $\alpha_k \leftarrow \Gamma(1-1/k)^{-k}$ `// normalizing factor, for large` k`:` $\alpha_k \approx \exp(-\gamma)$
  return $Z = \exp(\frac{1}{k}\sum_i X_i) \cdot \alpha_k$

▶ **Theorem 6.** *Applied to a stream of n distinct elements, Algorithm 1 outputs Z such that $\mathbb{E}[Z] = n$ and if $k > 2$ then $\mathrm{Var}[Z] = \frac{n^2}{k}\left(\frac{\pi^2}{6} + O(k^{-1})\right)$. It uses k real-value registers and spends $\mathcal{O}(k)$ operations per single processed element of the input.*

Proof. We analyze the Algorithm 1 after processing a stream of n distinct elements. For each X_i, its value is a maximum of n random variables drawn from $\mathsf{Gumbel}(0)$ distribution, so by Property 3 we have that $X_i \sim \mathsf{Gumbel}(\ln n)$. Hence, $X_i = G_i + \ln n$ where all G_i are identically distributed according to the $\mathsf{Gumbel}(0)$. Moreover, repeated occurrences of the elements do not change the state of the algorithm. Denoting $G = \sum_i G_i$, we get

$$\mathbb{E}[Z] = \Gamma(1-1/k)^{-k}\,\mathbb{E}[\exp(\ln n + G/k)] = n\Gamma(1-1/k)^{-k}\Gamma(1-1/k)^{k} = n$$

and

$$\mathrm{Var}[Z] = n^2\Gamma(1-1/k)^{-2k}\,\mathrm{Var}[\exp(G/k)] = n^2\left(\frac{\Gamma(1-2/k)^k}{\Gamma(1-1/k)^{2k}} - 1\right).$$ ◀

4.1 Stochastic averaging

We refine the Algorithm 1 by adding stochastic averaging. Application of the technique is straightforward, but for technical reasons we need to take care of the initialization of the registers – since the expected value of X_i is the logarithm of the number of the elements assigned to the i-th register, we don't want any of the registers to be empty at the and. Therefore, at the beginning we feed each of them with an artificial random element.

Algorithm 2 Cardinality estimation using Gumbel distribution and stochastic averaging.

Procedure *Init()*
  pick $h : U \to \{1, \dots, k\}$ and $r : U \to [0, 1]$ as independent hash functions
  for $1 \le i \le m$ **do**
    $X_i \leftarrow -\ln(-\ln u_i)$ where u_i is picked uniformly from $[0, 1]$. `// Gumbel(0) RV`

Procedure *Update*(x)
  $t \leftarrow h(x)$
  $v \leftarrow -\ln(-\ln r(x))$ `// Gumbel(0) RV`
  $X_t \leftarrow \max(v, X_t)$

Procedure *GeometricEstimate()*
  $\alpha_k \leftarrow \Gamma(1-1/k)^{-k}$ `// normalizing factor, for large` k: $\alpha_k \approx \exp(-\gamma)$
  return $Z = k \cdot \exp(\frac{1}{k}\sum_i X_i) \cdot \alpha_k$

▶ **Theorem 7.** *Applied to a stream of n distinct elements, Algorithm 2 outputs Z such that if $k > 1$ then $n\frac{k}{k+1} \le \mathbb{E}[Z] \le n + k$ and if $k > 2$ then $\mathrm{Var}[Z] \le 3.645\frac{n^2}{k} + \mathcal{O}(n^2/k^2 + k^2)$. It uses k real-value registers and spends constant number of operations per single processed element of the input.*

Proof. We analyze Algorithm 2 after processing stream S of n distinct elements. Let $n_1, \dots, n_k$ be the respective numbers of unique items hashed by h into registers $\{1, \dots, k\}$ respectively. It follows that $n_1, \dots, n_k \sim \mathsf{Multinomial}(n; \frac{1}{k}, \dots, \frac{1}{k})$. For each X_i, its value is a maximum of $n_i + 1$ random variables drawn from the $\mathsf{Gumbel}(0)$ distribution (taking into account n_i updates to its value and the initialization). Thus, conditioned on the specific values of $n_1, \ldots, n_k$, we have that X_i follows the Gumbel distribution – more specifically $X_i | n_1, \ldots, n_k \sim \mathsf{Gumbel}(\ln(n_i + 1))$. Let us denote $G_i = X_i - \ln(n_i + 1)$, $G = \sum_i G_i$ and $Y = \sum_i \ln(n_i + 1)$. We observe that G_i's are independent random variables distributed according to $\mathsf{Gumbel}(0)$ (and independent from Y).

We now have

$$Z = k\Gamma(1 - 1/k)^{-k} \exp(Y/k) \exp(G/k).$$

Since $\mathbb{E}[\exp(G/k)] = \Gamma(1 - 1/k)^k$ and G and Y are independent, using Lemma 8 we get

$$\mathbb{E}[Z] = k\,\mathbb{E}[\exp(Y/k)] \geq n \cdot \frac{k}{k+1}$$

and

$$\mathbb{E}[Z] = k\,\mathbb{E}[\exp(Y/k)] \leq n + k.$$

Now, using $\mathrm{Var}[AB] = \mathrm{Var}[A]\,\mathbb{E}[B^2] + \mathbb{E}[A]^2\,\mathrm{Var}[B]$ identity for independent random variables and Lemma 8 we can bound

$$\begin{aligned}
\mathrm{Var}[Z] &= k^2\Gamma(1-1/k)^{-2k}\left(\mathrm{Var}[\exp(Y/k)]\,\mathbb{E}[\exp(G/k)^2] + \mathbb{E}[\exp(Y/k)]^2\,\mathrm{Var}[\exp(G/k)]\right)\\
&\leq (k^2 + 2n^2/k + O(n^2/k^2))\frac{\Gamma(1-2/k)^k}{\Gamma(1-1/k)^{2k}} + (n+k)^2(\frac{\Gamma(1-2/k)^k}{\Gamma(1-1/k)^{2k}} - 1)\\
&= (k^2 + 2n^2/k + O(n^2/k^2))(1 + O(k^{-1})) + (n+k)^2(\frac{\pi^2}{6k} + O(k^{-2})),
\end{aligned}$$

and the claim follows. ◀

▶ **Lemma 8.** *Let $n_1, \ldots, n_k \sim \mathsf{Multinomial}(n; 1/k, \ldots, 1/k)$ and let $T = \sqrt[k]{\prod_i (n_i + 1)} = \exp(\frac{1}{k}\sum_i \ln(n_i + 1))$. Then there is $n/(k+1) \leq \mathbb{E}[T] \leq n/k + 1$ and $\mathrm{Var}[T] \leq 1 + 2n^2/k^3 + \mathcal{O}(n^2/k^4)$.*

Proof. Denote $Y = \sum_i \ln(n_i + 1)$. By Lemma 9 bound we have

$$\begin{aligned}
\mathbb{E}[\exp(Y/k)] &\geq \int_0^\infty \exp(\ln(n/k) - t/k)e^{-t}dt\\
&= n/k \int_0^\infty \exp(-\frac{k+1}{k}t)dt\\
&= n/(k+1).
\end{aligned}$$

By concavity of a logarithm we have $Y = \sum_i \ln(n_i + 1) \leq k\ln(n/k + 1)$, so $\exp(Y/k) \leq n/k + 1$. Finally, by Lemma 9 bound we get

$$\begin{aligned}
\mathrm{Var}[\exp(Y/k)] &\leq \mathbb{E}[(\exp(Y/k) - n/k)^2]\\
&\leq ((n/k+1) - n/k)^2 + \frac{n^2}{k^2}\int_0^\infty (1 - e^{-t/k})^2 e^{-t}dt\\
&= 1 + \frac{n^2}{k^2}\left(1 - 2\frac{k}{k+1} + \frac{k}{k+2}\right).
\end{aligned}$$ ◀

▶ **Lemma 9.** *Let $n_1, \ldots, n_k \sim \mathsf{Multinomial}(n; 1/k, \ldots, 1/k)$ and let $Y = \sum_i \ln(n_i + 1)$. Then $Y \geq k\ln(n/k) - t$ with probability at least $1 - e^{-t}$.*

Proof. Consider $\mathbb{E}[e^{-Y}]$. We have

$$\begin{aligned}
\mathop{\mathbb{E}}_{\substack{n_1,..,n_k\sim\\ \text{Multinomial}}}[e^{-Y}] &= \mathop{\mathbb{E}}_{\substack{n_1,..,n_k\sim\\ \text{Multinomial}}}\left[\prod_i \frac{1}{n_i+1}\right] \\
&= \sum_{i_1+\ldots+i_k=n} \Pr[n_1 = i_1 \wedge \ldots \wedge n_k = i_k] \prod_i \frac{1}{i_i+1} \\
&= \sum_{i_1+\ldots+i_k=n} k^{-n}\binom{n}{i_1,\ldots,i_k}\prod_i \frac{1}{i_i+1} \\
&= k^{-n} \sum_{i_1+\ldots+i_k=n} \frac{n!}{(i_1+1)!\cdot\ldots\cdot(i_k+1)!} \\
&= k^{-n} \sum_{i_1+\ldots+i_k=n} \binom{n+k}{i_1+1,\ldots,i_k+1}\frac{n!}{(n+k)!} \\
&\le k^{-n}k^{n+k}\frac{n!}{(n+k)!} \\
&\le \left(\frac{k}{n}\right)^k.
\end{aligned}$$

Thus, for any $t > 0$, by Markov's inequality

$$\begin{aligned}
\Pr[Y \le k\ln(n/k) - t] &= \Pr[e^{-Y} \ge e^{t-k\ln(n/k)}] \\
&\le \Pr[e^{-Y} \ge e^t \cdot \mathbb{E}[e^{-Y}]] \\
&\le e^{-t}.
\end{aligned}$$

◀

4.2 Discretization

Presented sketches use k real-value registers, which is in disadvantage when compared with LogLog and HyperLogLog, where only k integers are used, each taking $\mathcal{O}(\log\log n)$ bits. We now discuss how to reduce the memory footprint of the algorithms. This section exemplifies the usefulness of Gumbel distributions. In particular, this is a family of the limit distributions where *additive error* of registers corresponds to *multiplicative error* of estimation.

Simple rounding

First, we note that rounding the registers to nearest multiplicity of ε for some $\varepsilon > 0$ introduces at most $\exp(\varepsilon) = 1 + \varepsilon + \mathcal{O}(\varepsilon^2)$ multiplicative distortion in the estimation procedure GeometricEstimate() from both Algorithm 1 and 2. For example, for 1, we have, assuming X'_i are rounded registers: $|X'_i - X_i| \le \varepsilon$, and so for $Z' = \alpha_k \exp(\frac{1}{k}\sum_i X'_i)$ there is $\frac{Z'}{Z} = \exp(\frac{1}{k}\sum_i(X'_i - X_i))$, so $\exp(-\varepsilon) \le \frac{Z'}{Z} \le \exp(\varepsilon)$. Since each register stores w.h.p. values of magnitude $2\log n$, it can be implemented on integer registers using $\mathcal{O}(\log\frac{\log n}{\varepsilon}) = \mathcal{O}(\log\log n + \log\varepsilon^{-1})$ bits.

Randomized rounding

We now show how to eliminate the $\log\varepsilon^{-1}$ term. We define the following *shift-rounding*, for shift value $c \in [0, 1)$:

$$f_c(x) \stackrel{\text{def}}{=} \lfloor x + c\rfloor - c.$$

We note two key properties:

1. shift-rounding commutes with maximum, that is, for any $x_1, \ldots, x_k$, we have $\max(f_c(x_1), \ldots, f_c(x_k)) = f_c(\max(x_1, \ldots, x_k))$,
2. If $c \sim U[0,1]$, then $f_c(x) \sim U[x-1, x]$, where $U[a,b]$ denotes uniform distribution on range $[a,b]$.

We thus show how to adapt the Algorithm 2 using shift-rounding.

The analysis of Algorithm 3 comes from following invariant: if Algorithms 3 and 2 are run side-by-side on the same input stream, at any given moment there is $X'_i = f_{c_i}(X_i)$. Thus, we have the following $X'_i \sim \mathsf{Gumbel}(\ln(n_i+1)) - U[0,1] = \ln(n_i+1) + \mathsf{Gumbel}(0) - U[0,1]$.

Algorithm 3 Algorithm 2 with shift-rounding.

Procedure *Init()*
  pick $h : U \to \{1, \ldots, k\}$ and $r : U \to [0,1]$ as independent hash functions
  for $1 \le i \le m$ **do**
    c_i is picked uniformly from $[0,1]$
    $X'_i \leftarrow \lfloor -\ln(-\ln u_i) + c_i \rfloor - c_i$
    where u_i is picked uniformly from $[0,1]$. `// Gumbel(0) RV + randomized rounding`
Procedure *Update(x)*
  $t \leftarrow h(x)$
  $v \leftarrow \lfloor -\ln(-\ln h(x)) + u_i \rfloor - u_i$
  $X'_t \leftarrow \max(v, X'_t)$
Procedure *GeometricEstimate()*
  $\alpha'_k \leftarrow \Gamma(1-1/k)^{-k}(1-\exp(-1/k))^{-k}k^{-k}$ `// normalizing factor, for large` k: $\alpha'_k \approx \exp(1/2-\gamma)$
  return $Z = k \cdot \exp(\frac{1}{k}\sum_i X'_i) \cdot \alpha'_k$

Additionally, X'_i are independent as X_i were independent. We observe that for $X' \sim \mathsf{Gumbel}(0) - U[0,1]$, there is $\mathbb{E}[\exp(X'/k)] = \Gamma(1-1/k)(1-\exp(-1/k))k$, so we have equivalents of Lemma 5 in the following sense: $\mathbb{E}[\exp(\frac{1}{k}\sum_i G'_i)] = \Gamma(1-1/k)^k(1-\exp(-1/k))^k k^k \approx \exp(\gamma - 1/2)(1 + (\frac{\pi^2}{12} + \frac{1}{6})\frac{1}{k} + \mathcal{O}(k^{-2}))$, and $\mathrm{Var}[\exp(\frac{1}{k}\sum_i G'_i)] \approx \exp(2\gamma-1)((\frac{\pi^2}{6} + \frac{1}{3})\frac{1}{k} + \mathcal{O}(k^{-2}))$.

Thus an equivalent of Theorem 7 applies to Algorithm 3 with slightly worse constants.

▶ **Theorem 10.** *Applied to a stream of n distinct elements, Algorithm 3 outputs Z such that if $k > 1$ then $n\frac{k}{k+1} \le \mathbb{E}[Z] \le n + k$ and if $k > 2$ then $\mathrm{Var}[Z] \le 3.98\frac{n^2}{k} + \mathcal{O}(n^2/k^2 + k^2)$. It uses k integer registers of size $\mathcal{O}(\log\log n)$ bits each and spends constant number of operations per single processed element of the input.*

We note that each X'_i takes values only from set $\mathbb{Z} - c_i$ of magnitude at most $2\log n$, it can be stored using $\mathcal{O}(\log\log n)$ bits. Values of c_i do not need to be stored explicitly, as those can be extracted by picking a hash function $c : \{1, \ldots, k\} \to [0,1]$ and setting $c_i = c(i)$.

References

1 Algebird HyperLogLog implementation. Accessed: 2022-04-21. URL: `https://twitter.github.io/algebird/datatypes/approx/hyperloglog.html`.

2 Counting uniques faster in BigQuery with HyperLogLog++. Accessed: 2022-04-21, URL: `https://cloud.google.com/blog/products/gcp/counting-uniques-faster-in-bigquery-with-hyperloglog`.

3 HyperLogLog Sketch. Accessed: 2022-04-21. URL: `https://datasketches.apache.org/docs/HLL/HLL.html`.

4 Presto HyperLogLog function. Accessed: 2022-04-21. URL: `https://prestodb.github.io/docs/current/functions/hyperloglog.html`.

5 Redis PFCOUNT command. Accessed: 2022-04-21. URL: `https://redis.io/commands/pfcount`.

6 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29, 1996. `doi:10.1145/237814.237823`.

7 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *RANDOM 2002*, pages 1–10, 2002. `doi:10.1007/3-540-45726-7_1`.

8 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA 2002*, pages 623–632. ACM/SIAM, 2002. URL: `http://dl.acm.org/citation.cfm?id=545381.545464`.

9 Kevin Beyer, Rainer Gemulla, Peter J Haas, Berthold Reinwald, and Yannis Sismanis. Distinct-value synopses for multiset operations. *Communications of the ACM*, 52(10):87–95, 2009.

10 Joshua Brody and Amit Chakrabarti. A multi-round communication lower bound for gap hamming and some consequences. In *CCC 2009*, pages 358–368, 2009. `doi:10.1109/CCC.2009.31`.

11 Jarosław Błasiok. Optimal streaming and tracking distinct elements with high probability. In *SODA 2018*, pages 2432–2448, 2018. `doi:10.1137/1.9781611975031.156`.

12 Aiyou Chen, Jin Cao, Larry Shepp, and Tuan Nguyen. Distinct counting with a self-learning bitmap. *Journal of the American Statistical Association*, 106(495):879–890, 2011.

13 Peter Clifford and Ioana A Cosma. A statistical analysis of probabilistic counting algorithms. *Scandinavian Journal of Statistics*, 39(1):1–14, 2012.

14 Edith Cohen. All-distances sketches, revisited: Hip estimators for massive graphs analysis. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2320–2334, 2015.

15 Laurens De Haan and Ana Ferreira. *Extreme value theory: an introduction*. Springer Science & Business Media, 2007.

16 Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities (extended abstract). In *ESA 2003*, pages 605–617, 2003. `doi:10.1007/978-3-540-39658-1_55`.

17 Otmar Ertl. New cardinality estimation algorithms for hyperloglog sketches. *CoRR*, abs/1702.01284, 2017. `arXiv:1702.01284`.

18 Cristian Estan, George Varghese, and Michael E. Fisk. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Trans. Netw.*, 14(5):925–937, 2006. `doi:10.1145/1217709`.

19 Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, pages 137–156. Discrete Mathematics and Theoretical Computer Science, 2007.

20 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985. `doi:10.1016/0022-0000(85)90041-8`.

21 Lucas Gerin and Philippe Chassaing. Efficient estimation of the cardinality of large data sets. *Discrete Mathematics & Theoretical Computer Science*, 2006.

22 Phillip B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB 2001*, pages 541–550, 2001. URL: `http://www.vldb.org/conf/2001/P541.pdf`.

23 Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *SPAA 2001*, pages 281–291, 2001. `doi:10.1145/378580.378687`.

24 Frédéric Giroire. Order statistics and estimating cardinalities of massive data sets. *Discret. Appl. Math.*, 157(2):406–427, 2009. `doi:10.1016/j.dam.2008.06.020`.

25 Emil Julius Gumbel. Les valeurs extrêmes des distributions statistiques. In *Annales de l'Institut Henri Poincaré*, volume 5(2), pages 115–158, 1935.

26 Stefan Heule, Marc Nunkesser, and Alexander Hall. Hyperloglog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. In *EDBT 2013*, pages 683–692, 2013. `doi:10.1145/2452376.2452456`.

27 Piotr Indyk and David P. Woodruff. Tight lower bounds for the distinct elements problem. In *FOCS 2003*, pages 283–288, 2003. `doi:10.1109/SFCS.2003.1238202`.

28 T. S. Jayram and David P. Woodruff. Optimal bounds for johnson-lindenstrauss transforms and streaming problems with sub-constant error. In *SODA 2011*, pages 1–10, 2011. `doi:10.1137/1.9781611973082.1`.

29 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *PODS 2010*, pages 41–52, 2010. `doi:10.1145/1807085.1807094`.

30 Jérémie Lumbroso. An optimal cardinality estimation algorithm based on order statistics and its full analysis. *Discrete Mathematics & Theoretical Computer Science*, 2010.

31 Seth Pettie and Dingyu Wang. Information theoretic limits of cardinality estimation: Fisher meets shannon. In *STOC 2021*, pages 556–569. ACM, 2021.

32 Seth Pettie, Dingyu Wang, and Longhui Yin. Non-mergeable sketching for cardinality estimation. In *ICALP 2021*, volume 198 of *LIPIcs*, pages 104:1–104:20, 2021.

33 Wojciech Szpankowski. *Average case analysis of algorithms on sequences*, volume 50. John Wiley & Sons, 2011.

34 Daniel Ting. Streamed approximate counting of distinct elements: beating optimal batch methods. In *KDD 2014*, pages 442–451. ACM, 2014. `doi:10.1145/2623330.2623669`.

35 Alfredo Viola, Conrado Martínez, Jérémie Lumbroso, and Ahmed Helmi. Data streams as random permutations: the distinct element problem. *Discrete Mathematics & Theoretical Computer Science*, 2012.

36 David P. Woodruff. Optimal space lower bounds for all frequency moments. In *SODA 2004*, pages 167–175, 2004. URL: `http://dl.acm.org/citation.cfm?id=982792.982817`.

37 Qingjun Xiao, You Zhou, and Shigang Chen. Better with fewer bits: Improving the performance of cardinality estimation of large data streams. In *INFOCOM 2017*, pages 1–9, 2017.